ARPA ORDER NO.: 3460/3681
9P10 Information Processing Techniques

N-1158-1-ARPA

May 1979

DESIGN OF A RULE-ORIENTED SYSTEM FOR IMPLEMENTING EXPERTISE

D. A. Waterman, R. H. Anderson, Frederick Hayes-Roth, Philip Klahr,
Gary Martins, Stanley J. Rosenschein

A Rand Note

prepared for the

DEFENSE ADVANCED RESEARCH PROJECTS AGENCY

Rand
SANTA MONICA, CA. 90406

The research described in this report was sponsored by the Defense Advanced Research Projects Agency under Contract No. MDA903-78-C-0029.

Published by The Rand Corporation

ARPA ORDER NO.: 3460/3681
9P10 Information Processing Techniques

N-1158-1-ARPA

May 1979

DESIGN OF A RULE-ORIENTED SYSTEM FOR IMPLEMENTING EXPERTISE

D. A. Waterman, R. H. Anderson, Frederick Hayes-Roth, Philip Klahr,
Gary Martins, Stanley J. Rosenschein

A Rand Note

prepared for the

DEFENSE ADVANCED RESEARCH PROJECTS AGENCY

Rand
SANTA MONICA, CA. 90406

PREFACE

This Note describes the preliminary design of a Rule-Oriented System for Implementing Expertise (ROSIE). This system is intended as a tool for model builders seeking to apply expert knowledge to the analysis of problems and to the evaluation of solutions in complex domains, especially domains for which useful analytic models are unavailable.

This preliminary design--the result of a six-month design exercise--formed the basis of a proposal for implementation of the software system submitted to the Information Processing Techniques Office of the Defense Advanced Research Projects Agency.

The Note is being distributed to promote discussion and exchange of views with colleagues interested in rule-directed systems for heuristic modeling. It is intended for a technical audience; basic knowledge of the architecture of rule-based systems is assumed.

SUMMARY

The preliminary design has been completed of a modeling system that will enable experts and end-users alike to participate directly in the creation of interesting applications systems. ROSIE has been designed to be a flexible system capable of processing large quantities of information efficiently and effectively. In addition, it is able to facilitate interaction with the external world and is implementable within a short time period.

ROSIE is flexible and friendly, i.e., easy to modify, use and understand. This is accomplished by making ROSIE models rule-based and by providing the user with a support package that facilitates his use of the system. The rule syntax of ROSIE is similar to RITA: IF-THEN rules in an English-like framework. However, rule semantics have been expanded to facilitate iteration through a data set and to provide an abstraction and aggregation hierarchy mechanism. We have also introduced an event-driven monitor capable of testing when expressions become true. This permits the user to notice when things are changing and simplifies implementing alerts or other kinds of change detecting processes.

To handle the problem of processing large amounts of information we have modularized the rule and data elements so individual modules can be accessed and executed independently. This provides a means for maintaining only the currently active

and perhaps relevant modules in core at any one time. The mechanism for achieving modularity relies on the concepts of partitioning and activation. The user partitions his rules and data into separate sets based on his expectations regarding their interdependencies. Rule and data sets are activated, i.e., permitted to interact to cause rules to fire, only when deemed relevant by the user or the ROSIE monitor.

The support package in ROSIE includes many features for assisting the user, all built around the notion that rules are simply another type of data element that may be accessed and manipulated by rules. Editing facilities are rule-based and thus may be extended or modified by the user. The user may construct auxiliary rule sets that assist him in determining rule correctness by examining the main rule set, looking for important similarities or differences in rules. A sophisticated explanation facility is included that traces the operation of the system at various levels, providing a way to justify system inferences and debug faulty rule sets. Reasoning in the presence of uncertainty is handled by permitting the user to assign weights or "certainty factors" to rules and data. The user can then specify a certainty range, and only rules and data with certainty factors in that range will be used in the calculation.

CONTENTS

I. INTRODUCTION

This Note describes the preliminary design of a Rule-Oriented System for Implementing Expertise (ROSIE). This system will serve as a tool for model builders seeking to apply expert knowledge to the analysis of problems and the evaluation of solutions in complex domains, especially domains for which useful analytic models are unavailable. Its basic simplicity, together with powerful user-support features, will encourage enterprising users to take the lead in developing innovative models to serve their own mission areas.

Computer systems that faithfully incorporate human judgmental expertise offer substantial advantages to the military, especially if they can be built with reasonable effort. They promise to make such expertise widely sharable, helping to relieve the demand for highly trained and experienced operational personnel. Additionally, a single system may incorporate the expertise of many contributors, resulting in a net improvement in the overall mission performance of the system. In most cases, ROSIE programs are expected to serve as aids to--not replacements for--human decisionmakers, whose performance they will sharpen and stabilize.

Potential applications for systems of this kind abound in the civilian and military worlds. Military applications are anticipated in areas such as:

tactical: ops planning, experiments, gaming

personnel: training, testing, practice

mil ops: situation analysis, plan evaluation

logistics: basing, staging

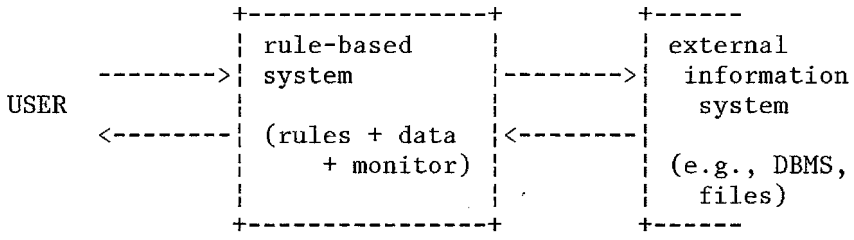
maintenance: cycle planning, policy evaluation

Models built within the ROSIE system may constitute simulations in the application domain. Control in these models is data-directed (Hayes-Roth, Waterman, & Lenat, 1978; Waterman, 1978a, 1978b); that is, actions are specified by sets of rules.

For readers not familiar with rule-based systems and prior Rand R&D in this area, the following background information provides some additional context. A rule-based system can be thought of as having three components: a set of rules of the form

IF <conditions> THEN <actions>;

a data base against which the rule conditions are tested, and which is altered by the execution of rules' actions; and a monitor program that contains logic regarding the order in which rules are to applied, what to do in case more than one rule applies (i.e., has true conditions) at one time, and so forth. The rule-based system may be situated between the user and other external systems as shown below.



That is, the rule-based system is capable of interacting with the user (e.g., to obtain advice, to explain its behavior upon request) and also communicating with one or more external information systems (which might be contained within the same host computer, or accessed via data networks) to obtain needed information in the course of its calculations. Within this general system architecture, the rule-based system might play several different roles:

- o A decision aid or planning aid for the user, containing a number of rules ("heuristics") that guide its deliberations in generating plans or proposed decisions. In this role, the logic within the rules and data of the rule-based system is of paramount interest, with the rule-based system possibly calling upon external information systems for needed data;

- o A flexible interface to external information systems. In this case, the user's primary interest is in the external system, but he prefers to interact with that system through a tailored (rule-based) interface capable of mapping user requests into an interactive dialog that extracts needed information from the external system. Here the rule-based system often acts as a "surrogate user", dealing with the external system as if it were a human user of that system; in this manner, no changes are needed in the external system in order to obtain the advantages of this tailored interface.

ROSIE is a system that allows rules + data + monitors to be

developed that will become a total "rule-based system" for any of the above uses. Its design is quite heavily influenced by earlier experience with a similar, but much simpler, rule-based system developed by Rand called RITA (Anderson & Gillogly, 1976; Anderson et al, 1977). Readers desiring additional background information on the design philosophy involved in ROSIE's precursors and examples of the uses of rule-based systems are urged to consult the above references.

The ROSIE design exploits and integrates many current ideas in artificial intelligence research. The use of a rule-based language builds on previous work on MYCIN (Shortliffe, 1976) and other work in pattern-directed inference system design (Waterman & Hayes-Roth, 1978b). The event- or change-driven rule-invocation strategies are based on the use of demons in PLANNER and ARS (Hewitt, 1971, 1972; Stallman & Sussman, 1976) and similar schemes for speech understanding (Hayes-Roth & Mostow, 1975). We have borrowed the idea of a hierarchical data structure capable of supporting abstraction and inheritance from the work on units and frames (Minsky, 1975; Martin et al., 1977; Lenat, 1976, 1977; Lenat & Harris, 1978; Winograd, 1975; Bobrow & Winograd, 1977; Charniak, 1975; Havens, 1978). Many of the ideas related to the use of rules as data and the inclusion of elaborate user support features were inspired by INTERLISP (Teitelman, 1974). We intend to retain the positive user-oriented features of these languages while incorporating new features that simplify the handling of rich and complex domain

descriptions. Major features include:

- o hierarchic structures on data elements and rules, to support abstraction in the models
- o user selection of existence-driven or event-driven rule-invocation strategies
- o user control of rule iteration
- o user control of rule and data activation, including a "rule-subroutining" capability
- o user support tools

In the design of the ROSIE system, our primary aim has been to support the creation of realistic models. We know from experience with RITA and other rule-based systems, that realistic modeling implies fairly large sets of model elements: rules and data elements. We also know that to be useful to end-users (i.e., people with expertise in some significant problem domain, but lacking expertise in conventional computer programming), a powerful system must be easy to learn and use (Waterman, 1977; Waterman & Jenkins, 1977). Hence, the main requirements on the ROSIE design are:

- o efficient and effective handling of large rule and data sets for realistic modeling
- o a "friendly" user environment that facilitates both system building and use
- o system flexibility and modifiability to allow exploration of implementation alternatives
- o implementation within a relatively short time period to support near-term applications

Our choice of an implementation environment for ROSIE was largely determined by these criteria. Embedding the prototype system in INTERLISP will permit fast implementation and allow a flexible approach to monitor strategies and other key system decisions. Running the ROSIE system on a PDP-10 class computer will give users the speed and memory capacity needed for building large models.

The following sections describe in greater detail the features of the ROSIE system. Section II discusses the ROSIE design requirements, relating them to the current design. Sections III and IV describe data and rule specifications and Section V concludes with a discussion of the user support environment.

II. SYSTEM DESIGN -- AN OVERVIEW

LARGE RULE/DATA SETS

Vast amounts of information are needed to reach decisions in complex application areas. To handle this problem we have modularized the rules and elements so individual modules can be accessed and executed independently. This provides a means for maintaining only the currently active and perhaps relevant modules in core at any one time. The mechanism for achieving modularity relies on the concepts of partitioning, activation, and abstraction. The user partitions his rules and data into separate sets based on his expectations regarding their interdependencies. Rule and data sets are activated, i.e., permitted to interact to cause rules to fire, only when deemed relevant by the user or the ROSIE monitor.

The user is able to handle many different kinds of rules at different levels of generality through abstraction, i.e., organizing the elements so that the "INSTANCE" relations between very general and very specific elements are made explicit. General rules apply to categories of data types called concepts. We have used data abstraction in order to make it easy to write rules that apply to all instances of general concepts. This works by allowing elements that represent low-level or very specific concepts (e.g., a carrier) to inherit attributes specified by higher-level or more general concepts (e.g., naval platform) of which they are instances. Thus, a rule that checks

to see if "carriers" have some attribute x will be satisfied if either the proper value of the attribute is associated with "carrier" or is associated with a more general concept of "carrier" such as "naval platform."

Similarly, aggregation is used to permit the user to access collections of elements using simple rules. Here the "member of" relation between aggregated elements and their constituent parts is made explicit. Questions about being a member of something are answered by finding the closure of all sets and subsets that are members of the element in question.

Finally, we will achieve a significant efficiency by allowing the rules to fire in response to events or changes in the data base. This use of an event-driven monitor also simplifies the rules, permitting the user to create rule sets that act as large collections of demons acting independently of one another. In addition, we envisage permitting the user to formulate different control rules, which we call monitor programs, that would be specially adapted to the efficient use of large sets of rules or searches of large data bases.

FRIENDLY SUPPORT ENVIRONMENT

A primary goal is to make the system familiar and friendly. By familar we mean non-radical, extending ideas already developed in other systems. For example, we have borrowed the idea of an English-like syntax from RITA and MYCIN, the concept of data abstraction hierarchies from a number of AI

programs (Minsky, 1975; Lenat, 1977; McCalla, 1978), and the idea of recognition nets to speed up the rule matching from the ACORN work (Hayes-Roth & Mostow, 1975). We understand these ideas quite well because they have been implemented either at Rand or elsewhere several times before. By friendly we mean a system that is easy to use and understand. We accomplish this in two ways--by designing the system around a simple rule syntax and by providing the user with a support package that facilitates his use of the system.

The rule syntax of ROSIE is quite similar to RITA: IF-THEN rules in an English-like framework. However, most of the awkwardness of RITA programming is gone. For example, in RITA it is difficult to write rules that look for a certain kind of pattern in the data and then perform a particular action to all instances of the data elements matching that pattern. It is difficult to write single rules that apply to classes of data elements. Also, at present the user often needs to specify the program state as a condition for rule firing and a change of state as an action in order to obtain sequential rule firings or prevent a single rule from firing repeatedly. To avoid these problems we have expanded rule semantics to facilitate iteration through a data set and have provided the abstraction and aggregation hierarchy mechanism. We have also introduced an event-driven monitor to allow expressions to be tested for their becoming true. Thus rules can detect if an expression is currently true but was not true on the last tested cycle and

cause appropriate action to be taken. This permits the user to notice when things are changing and simplifies implementing alerts or other kinds of change detecting processes.

The support package includes many features for assisting the user, all built around the notion that rules are simply another type of data element that may be accessed and manipulated by rules. Editing facilities in ROSIE are rule-based and thus may be extended or modified by the user. The user may construct auxiliary rule sets that assist him in determining rule correctness by examining the main rule set, looking for important similarities or differences in rules. A sophisticated explanation facility is included that traces the operation of the system at various levels, providing a way to justify system inferences and debug faulty rule sets. Reasoning in the presence of uncertainty is handled by permitting the user to assign weights or "certainty factors" to rules and data. The user can then specify a certainty range, and only rules and data with certainty factors in that range will be used in the calculation.

INTERACTION WITH THE EXTERNAL WORLD

One of the distinctive strengths of the RITA system, when compared to other existing production-rule programming systems, is the simplicity and power of its facilities for interaction with the external world. The ability of user models to affect the external world, and to be affected by external events, is important for many applications; it is indispensable for command

and control decision support models which require the monitoring and interpretation of real-world situations. An especially important application of a real-world interface is the driving of graphic and alphanumeric display systems by the user's model. The experience of many RITA users suggests that RITA's relation to the external world is the appropriate model to pursue for the new system.

This interaction is mediated through the exchange of character strings with the host operating system. The right-hand sides of rules influence the behavior of the host computing system just as would a user sitting at a terminal--by composing commands to the host system, and receiving messages from the host operating system in reply. For example:

```
IF users OF system IS NOT KNOWN
THEN SEND "systat" TO tenex
AND RECEIVE {ANYTHING FOLLOWED BY sys-prompt}
        AS users OF system;
```

```
WHEN THERE IS AN active-force [f]
        WHOSE astab IS NOT CURRENT
THEN FOR ALL CASES SEND astab-request OF [f] TO ladder
AND RECEIVE {ANYTHING FOLLOWED BY end-of-report}
        AS astab OF [f];
```

In this way, the user's rules can exercise all of the host system's capabilities, including network access (where available) to other systems. Such an approach, which takes advantage of all existing operating system facilities, pays off in two ways: it builds upon the user's knowledge of the host system, and it simplifies the implementation of ROSIE.

The exchange of character strings with the host system requires the existence of string analysis and composition mechanisms within ROSIE itself. These will resemble the highly successful RITA "pattern" and "concatenate" features.

MODIFIABILITY

It is important to make the system modifiable by the user to reflect his growing insight and expertise. Thus the monitor programs, the code controlling the way the rules make contact with the data base, are themselves formulated as rule-based programs. Two or three alternative rule-based monitors will be made available, although the sophisticated user will have the option of modifying or rewriting them himself. In this way we make almost all of the facilities of the system accessible to the user. Not only have we carried over the idea from LISP that "data equals program," but we have carried over the good ideas from INTERLISP that the system facilities themselves are written in the same formalism as the applications. Thus increasing the user's expertise in the applications program provides a capability for simultaneously increasing his expertise in the overall system.

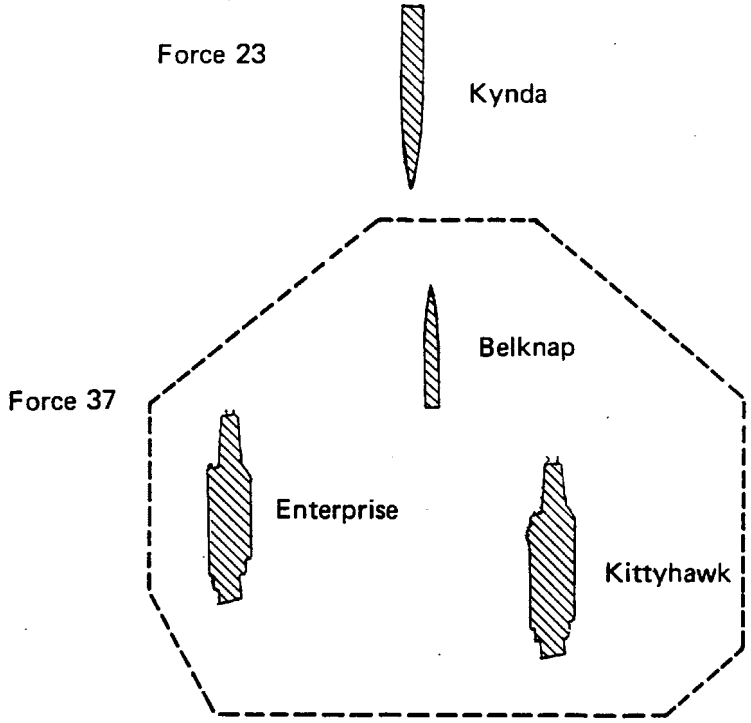


Fig. 1 — Naval Force Configuration

A typical ROSIE application is threat assessment, i.e., representing friendly and enemy forces is a manner that facilitates the recognition of an immediate or potential threat to one side or the other. Figure 1 is a simple illustration of a naval force configuration that is amenable to threat analysis. The problem is to develop a data base representing this configuration and rules describing how to calculate threats. This example will be used throughout the paper to assist in describing the design and use of ROSIE.

III. DATA SPECIFICATION

The basic system components are called knowledge elements. Knowledge elements represent cohesive pieces of knowledge such as events or the status of material objects, and the collection of these elements is called the knowledge base (sometimes referred to as the "data base" in other rule-based systems). Two types of elements exist: concepts and objects. A concept is an element that describes a class of data elements, e.g., plane, ship, battle, war. Classes of events may be represented as concepts, e.g., "losing a battle," while particular events may be represented as objects, e.g., "losing battle 34." Concepts have information associated with them that is representative of the class in general. For example, the concept "plane" might have the following associated information: it flies through the air, is self-propelled, etc. An object, on the other hand, represents a specific "real world" entity. It is usually a particular instantiation of some concept. For example, the data base might contain the concept "carrier" and an object "Enterprise," representing a particular carrier.

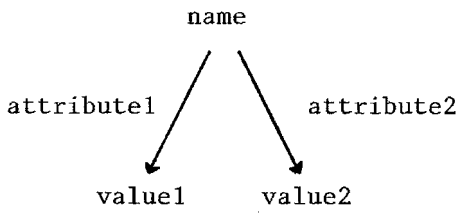
ELEMENT FORMS

An element is composed of a name with any number of associated attribute-value pairs. The name is a string of text that references or "names" the element, while the attributes are characteristics of the element that can have associated values.

The name-attribute-value triple can be represented either as statements or graphs. These are illustrated below.

The <attribute1> of the <name> is <value1>.

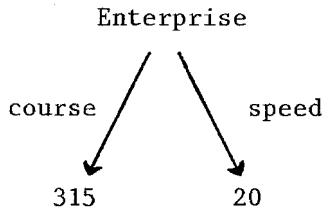
The <attribute2> of the <name> is <value2>.



These representations can be used to describe the object "Enterprise" from Figure 1 as shown below.

The course of the Enterprise is 315.

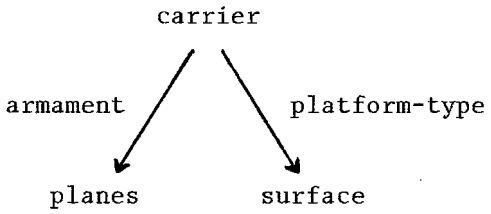
The speed of the Enterprise is 20.



A similar example for the concept whose name is "carrier" is shown below.

The armament of a carrier is planes.

The platform-type of a carrier is surface.

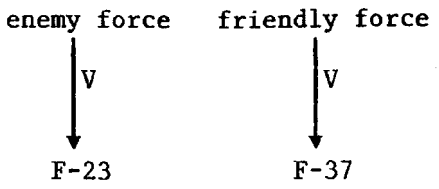


We will later show how to link these two representations into a coherent structure called an element hierarchy (see Figure 2).

It is also possible to associate values directly with names. The name has an implicit default attribute called "own value" (V). For example:

The enemy force is F-23.

The friendly force is F-37.

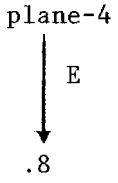


Two types of attributes are permitted in ROSIE, user-defined attributes and system attributes. The user-defined attributes are arbitrary words representing relations the user would like to define between the element's name and value. The system attributes are reserved words with special meaning to the ROSIE monitor. The four most important system attributes are "OWN VALUE" (V), "EXISTENCE" (E), "INSTANCE" (I), and "MEMBER" (M).

The "OWN VALUE" attribute (V) directly links a value to a name, thus providing a basic binding mechanism analogous to the assignment of values to identifiers in conventional programming languages. This attribute is untyped and can be set by a simple assignment statement, e.g., "SET enemy-force TO F-23" sets the V attribute of enemy-force to F-23. Evaluating an object consists of returning the value of its V attribute. The V attribute permits the construction of more compact, succinct element descriptions in many cases, e.g., using "the enemy force IS F-23" rather than "the current name OF the enemy force IS F-23."

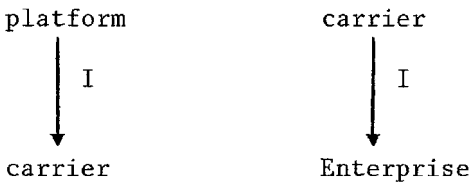
The "EXISTENCE" attribute (E) applies to objects and has a value representing the certainty that the object actually exists. For example, if a blip on a radar screen is interpreted as an enemy plane it may be important to include an estimate of

certainty that the plane does exist, as shown below.



Here the certainty that plane-4 exists is estimated as .8. If no information about the value of the E attribute is present in the knowledge base, it is assumed to be 1 (completely certain).

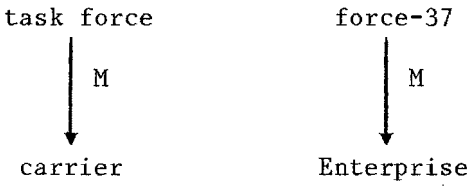
The "INSTANCE" attribute (I) is used to link concepts to other concepts or objects that are instances of or examples of the original concept. When the I attribute is used to form a name-attribute-value link the name is always a concept, the attribute is I, and the value is either an object or concept name. Examples are shown below.



The examples state that an instance of a platform is a carrier, and an instance of a carrier is the Enterprise. Since the value of an attribute can be an element name, the I link can be used to build complex nets, as discussed in the next section.

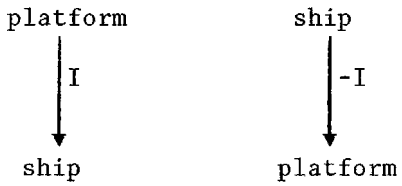
The "MEMBER" attribute (M) is used to link elements to other elements that represent components of the original element.

The M attribute always links concepts to other concepts, and objects to other objects (see below).



The examples state that "a member of a task force is a carrier," and "a member of force-37 is the Enterprise."

All system attributes have corresponding inverse attributes whose links are automatically defined when the original attribute is defined. Thus when the user states "an INSTANCE OF a platform IS a ship," or "ship IS an INSTANCE OF a platform" the following links are made between ship and platform:



indicating that the concept "platform" includes the special case "ship."

ELEMENT HIERARCHIES

There are two fundamental kinds of hierarchies that can be constructed in ROSIE, abstraction hierarchies and aggregation hierarchies. The abstraction hierarchy is defined by "INSTANCE" links between high-level concepts and lower-level concepts or objects. The links from higher concepts to lower concepts are traversed via the reserved attribute "INSTANCE" (I), while upward links are traversed via the reserved attribute "IS A" (-I). Hierarchies of this special kind can be created by type declarations as well as by direct manipulation of the reserved attributes. Examples of type declarations of the intended kind are:

- EVERY ship IS a platform
- EVERY OBJECT WHOSE armament IS planes IS a carrier
- platforms INCLUDE surface craft AND aircraft
- surface craft INCLUDE ships AND submarines
- ships INCLUDE carriers, destroyers AND cruisers
- carriers INCLUDE the Enterprise AND the Kittyhawk

The abstraction hierarchy is useful for expressing permanent type-token relationships among the objects in the universe. This hierarchy is a network of elements connected by I links, e.g.,

```
        I       I       I
e1 ---> e2 ---> e3 ---> e4.
```

Each element in the hierarchy inherits the user-defined attributes and values above it (going against the arrows) in the network. If the same attribute can be accessed more than once during upward traversal through the links, the lowest (closest) attribute-value pair is inherited. In Figure 2, the information that carriers carry planes and are surface vessels does not have to be stored repetitively with the objects "Enterprise" and "Kittyhawk." Instead, these objects "inherit" these values through upward traversal of the I-links. Thus a rule referring to a ship whose armament is planes would match both "Enterprise" and "Kittyhawk." This I-link inheritance can be suppressed by the addition of a "DON'T INHERIT" flag as an additional property of the attribute.

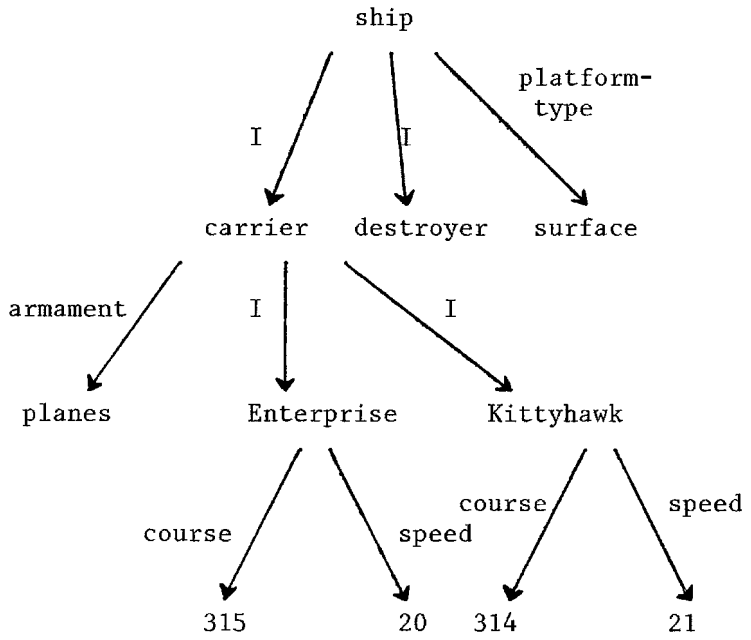


Figure 2. An Example of an Abstraction Hierarchy

Aggregation hierarchies can be built by connecting elements via the "MEMBER" or M link. They may be accessed by using "MEMBER" in the retrieval clause, e.g., "IF THERE IS a ship THAT IS A MEMBER OF a U.S. task force," or "IF THERE IS A MEMBER OF a U.S. task force WHOSE name IS Enterprise." When M-hierarchies and I-hierarchies intersect, the search for MEMBER will proceed appropriately through the I-hierarchy as well. There is no inheritance of attributes through the M-links.

Since the user may himself define new attribute types and link them to elements as desired, he has the potential for creating arbitrarily complex networks in the data base. These networks are created by explicit manipulation of attributes and

links, either by editing or by rule-directed manipulation of attributes. The only privileged monitor operation supported for these hierarchies is transitive traversal of these links during matching.

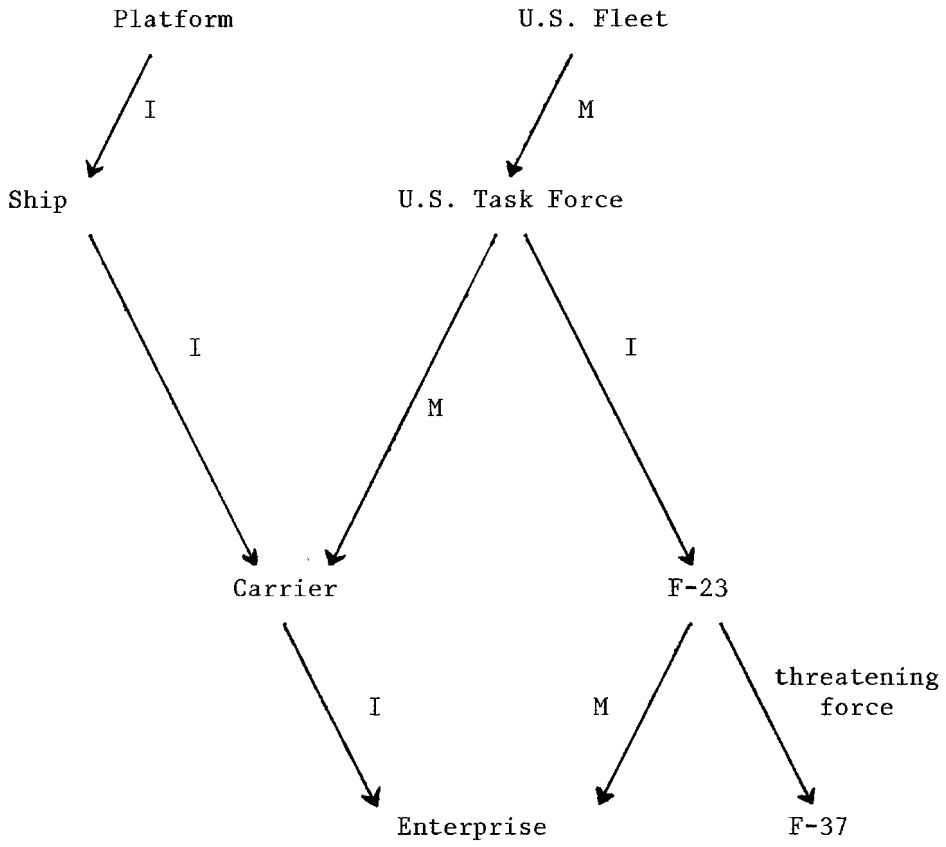
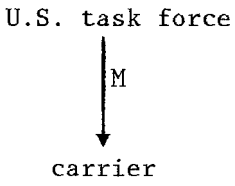


Figure 3. An Example of Abstraction and Aggregation Hierarchies

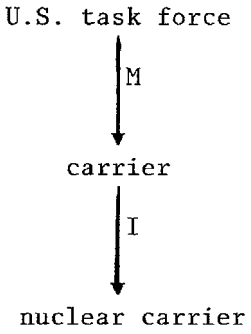
Objects can only be members of objects and concepts can only be members of concepts, but both objects and concepts can be instances of concepts. Figure 3 illustrates the use of I, M and user-defined attributes in a data hierarchy that partially

represents the configuration shown in Figure 1. System-defined attribute types are shown in upper case. Note that "threatening force" is a user-defined attribute name. The user (or system builder) defines his own attribute names and element names; the naming convention is completely arbitrary. Alternatively, "threatening force" could have been used as an element name.

Concepts linked to concepts through member-of relations mean that every instance of the higher level concept contains an instance of the lower level concept, for example,



means that every U.S. task force contains a carrier, but not that every carrier is a member of a U.S. task force. Links composed of member-instance pairs lead to "possible" or "could be" inferences, as shown below.



Here we may infer that a nuclear carrier could be a member of a

U.S. task force, but not that this is necessarily true. Whether or not we want to incorporate mechanisms to deal with inferences of this sort (and other similar ones) is still an open question.

An attribute has other information associated with it besides the value to which it is pointing. It has a data TYPE that can be number, list, string, element-name, or boolean; an INHERITIBILITY flag that determines whether or not it will be inherited via the I-links, and a CERTAINTY factor describing how certain it is that the attribute of the element has the given value.

There is a universal system concept (an implicit top node) that can have associated with it default attributes, attribute types, and values. All elements then inherit these properties. For example, if the system concept has the attribute LOCATION, with type LIST, then all elements in the system would have it.

IV. RULE SPECIFICATION

Rules are represented as conventional knowledge elements; hence they have attribute/value pairs associated with them and can be accessed and modified by other rules. The feature that distinguishes a rule element from a data element is the presence of a "condition" attribute representing the rule's left-hand and an "action" attribute representing the right-hand side. Shown below are other built-in attributes rules may have in addition to those the user may care to define. (Note that this list is not exhaustive.)

Name:	name of the rule (must be unique)
Condition:	the left-hand side of the rule
Action:	the right-hand side of the rule
Certainty:	certainty factor associated with the rule
Priority:	priority relative to other rules
Creator:	name of creator
Date:	creation date
Purpose:	explanation of rule purpose
Comments:	additional commentary regarding the rule

Since rules are simply another form of knowledge element, they are amenable to internal analysis by other rules and to inclusion in hierarchies within the system. There is no formal distinction between rules that manipulate rule elements and rules that manipulate data elements. This mechanism will be convenient and

useful for selecting from a very large set of rules and objects those that constitute interesting subsets for review, correctness checking, or activation.

The monitor that controls rule matching, selection and execution can be chosen by the user from a menu of available monitors. If none fits his specifications and he is an experienced programmer he will be able to modify existing monitors or write his own in ROSIE. The default monitor that is available is the ordered monitor. This monitor assumes that priorities have been assigned to each rule; these priorities are often based on the order the rules are entered into the system. A cycle in this system consists of selecting a rule that matches the data and executing it. The highest priority rule that currently matches is the one selected. Once the rule actions are executed (creating the possibility of new rules that match) the cycle starts again. This continues until no rules match or the action STOP is executed.

RULE FORMS

Rules fall into three categories: WHEN-THEN, IF-THEN, and DO. The WHEN-THEN rule cannot fire more than once for each distinct (set of) knowledge element(s) that matches its conditions or left-hand side (LHS). The only way it can fire again on the same element is when the matching value of the element has been changed. This is an example of an event-driven or demon-like rule. This rule has the form shown below.

```
WHEN <conditions> THEN <actions> {ELSE <actions>}

example:

WHEN THERE IS a ship WHOSE affiliation IS NOT KNOWN AND
the DISTANCE BETWEEN the ship AND the U.S. task force
IS LESS THAN 30 miles
THEN ADD the name OF the ship TO the potential threat list
     AND SEND the name OF the ship TO the USER
```

In the above example all ships with unknown affiliation and close proximity to the U.S. task force are added to the potential threat list. Each time the rule fires, one new ship is added, i.e., the rule must fire n times to add n ships to the list. After a ship's name is added to the list it is sent to the user. Because the rule fires only on knowledge base changes and only once for each data element no special mechanism is needed to keep the rule from being invoked continuously for the same knowledge elements.

The IF-THEN rule is analogous to the standard RITA rule. It is existence-driven; it fires repeatedly as long as the conditions are true, even if the elements matching its conditions have not been changed. The actions are executed once during each monitor cycle; repeated rule firings require repeated cycles. The form of this rule is shown below.

```
IF <conditions> THEN <actions> {ELSE <actions>}

example:

IF the state OF the system IS "compute relative threat"
THEN SET the state OF the system TO "set threat level"
   AND SET the relative threat OF the system TO (100
   * attack density OF the U.S. task force)/
   engagement density OF the U.S. task force
```

After the THEN actions have been performed and the conditions are no longer true, rule firing is terminated.

The DO rule is analogous to a RITA "immediate action." When used as part of a rule set, it behaves like a rule that is always true, e.g., "IF TRUE THEN <actions> and executes its actions each time it is tested by the monitor. When used alone it has the effect of a command and is executed as soon as it is read by the monitor. It has the form shown below.

DO <actions>

example:

```
DO ACTIVATE RULESET rs23
   AND ACTIVATE DATASET d15
```

This type of rule permits the user to effectively insert commands into his rule sets. This capability was found to be quite useful in the RITA system.

Rule actions can have the following forms. (Note that this list is not exhaustive.)

assignment:	SET <attribute> OF <name> TO <value>
list:	PUT <value> INTO <attribute> OF <name>
creation:	CREATE <item> (creates elements or attributes)
deletion:	DELETE <name> ¦ DELETE <attribute> OF <name>
I/O:	SEND, RECEIVE, OPEN, CLOSE, READFILE
termination:	STOP ¦ RETURN
rule:	WHEN-THEN, IF-THEN, or DO

activation: ACTIVATE <rulesets> ¦ DEACTIVATE <rulesets>

subroutine: CALL <rulesets>

The assignment, list, creation, deletion and I/O actions all correspond to useful actions available in RITA. The use of a rule as an action allows the user to create conditional expressions within the right-hand side (action side) of a rule. Experience with RITA has shown that this capability can significantly reduce the number of rules needed to express certain types of repetitive procedures. The activation and subroutine capabilities facilitate organizing the program in a modular form that is more efficient and easier to debug. More will be said about these capabilities in the next section.

Rule conditions have the form of a boolean expression with parentheses for disambiguation, e.g., A & (B v C) & -D. The two basic forms of the expression are:

<attribute> OF <name> IS <value>
THERE IS <name> WHOSE <attribute> IS <value>.

Again this list is not exhaustive, as there are many relations other than equality (e.g., greater than, less than, contains, between, etc.) needed to provide the user with a workable set of tools for rule construction.

INSTANCE SETS

When a rule's left-hand side is tested by the monitor an attempt is made to form the instance set for the rule's condition. The instance set of a boolean expression is the union of all ordered subsets of elements that match the expression, assuming automatic inheritance of attributes for more specific elements, i.e., those lower in the I-link hierarchy tree. This set is then used to instantiate rule variables. The methods used to create and use the instance set depend on the type of monitor being used and the type of rule being executed.

The SET action in "SET <attribute> OF <name> TO <value>" stores the new value at the highest level element in the instance set and deletes existing values at lower levels, unless they are flagged for no inheritance. (If necessary, cached values are updated.)

In the condition part of a rule the user may explicitly mention the type of knowledge element being sought. For example:

```
IF THERE IS a <name> WHOSE ...
IF THERE IS a CONCEPT <name> WHOSE ...
IF THERE IS an OBJECT <name> WHOSE ...
IF THERE IS an INSTANCE OF CONCEPT <name> WHOSE ...
```

In the "CONCEPT <name>" reference the instance set is the whole tree including the element referred to by <name> and all instances and abstractions under it.

The clause "a <name> WHOSE <attribute> IS <value>" causes a search for the attribute starting at the <name> element and

proceeding down the "INSTANCE" hierarchy. If the attribute has still not been found when the objects are reached, the search continues back up the tree above the original <name> node until either the attribute is found or the tree terminates.

CASE PHRASE

The rule "WHEN <conditions> THEN <actions>" executes its actions for one member of the instance set each time it is fired. If it is desired to execute the actions for all members of the instance set during a single rule firing, the rule must be reformulated as "WHEN <conditions> THEN FOR ALL CASES <actions>." The "FOR ALL CASES" phrase may be used with IF-THEN rules in the same manner. The example below illustrates the "FOR ALL CASES" phrase.

RULE 1: WHEN THERE IS a ship WHOSE speed IS LESS THAN 20 KNOTS
THEN SEND the name OF the ship TO the USER

RULE 1a: WHEN THERE IS a ship WHOSE speed IS LESS THAN 20 KNOTS
THEN FOR ALL CASES SEND the name OF the ship TO the USER

When rule 1 is tested against the knowledge base and found to be true it is executed only for the first ship in the knowledge base whose speed is less than 20 knots. Thus, other rules are tested and made available for execution before rule 1 necessarily has a chance to fire again for other ships with speeds less than 20 knots. Rule 1a, on the other hand, does not relinquish control to other rules until it has been executed for every ship in the knowledge base with speeds less than 20 knots. This permits the

user to write rules that can efficiently iterate through the data when so desired.

VARIABLES

Variables can be used in rule expressions to represent element names, attributes or values. The variable is identified by being enclosed in brackets, for example:

```
a ship [x] WHOSE length IS GREATER THAN 150
an ELEMENT [x] WHOSE ATTRIBUTE [y] IS VALUE [z]
IF  THERE IS an ELEMENT [x] THAT IS A MEMBER OF a fleet [y]
THEN SET the status OF [x] TO the status OF [y]
```

The binding of a variable takes place when the variable first occurs in the expression, typically after the defining term. The defining term can be "ELEMENT," "OBJECT," "CONCEPT," or a particular element, object, or concept. The default defining term is "ELEMENT." The scope of the binding is within one rule.

DATASETS/ACTIVATION

We allow named datasets and the ability to activate or deactivate them, e.g., "ACTIVATE classified-value-table." Since rules and data are treated alike the same activation mechanism is used for both, i.e., rulesets and their rules are just knowledge elements that can be (de)activated like data. Activation can be initiated either by user commands or by rule actions. Of course activating rules is quite different (in terms of effect) from activating data, since the monitor makes a clear distinction

between rules and data. Only rule elements can be executed whereas all elements, including other rules, can be matched against the left-hand sides of rules during condition testing.

There are two kinds of activation: global and local. Global activation involves defining a permanent operating context, i.e., the set of rules and data currently available for processing. The actions "ACTIVATE" and "DEACTIVATE" will be given the following meaning: "ACTIVATE alpha" means mark all the rules in the set alpha as accessible for current operations; "DEACTIVATE alpha" means mark all the rules as not accessible. This can be applied easily to both rules and objects without distinction. When elements are activated this way by a rule's action they are not available for processing until the execution of the rule has terminated.

Activation and deactivation of rules and data is handled uniformly by the actions shown below. Activation adds rules or data in the named set to the active set.

```
ACTIVATE RULESET <name>
ACTIVATE DATASET <name>
```

Deactivation removes rules or data from the active set.

```
DEACTIVATE RULESET <name>
DEACTIVATE DATASET <name>
```

Local activation involves defining a temporary operating context, i.e., a set of rules and data that are active only while the rule that activated them is still being executed. Thus local

activation is analogous to a subroutine call, and the action "USE" will be used to indicate this type of activation. Hence, "USE alpha" means that the rules in alpha become the current active set and all other rules in the system are marked as inaccessible until a "RETURN" action is executed in alpha. The effect of executing the return will be to reinstantiate the set of rules that existed prior to the use action. The distinction here is that there is a push and pop stacking mechanism that applies to "USE" and "RETURN," and does not apply to "ACTIVATE" and "DEACTIVATE." The form of the USE action is shown below.

USE RULESET <name>

There is an open question as to whether "USE" and "RETURN" should apply to data as well as rules. A second more fundamental issue concerns the passing of parameters via the "USE" action to a new rule set. The parameters to be passed should be subject to the push and pop mechanism, along with the set of active rules so that the "USE" mechanism can be used recursively.

Datasets can be defined by actions in rules, as illustrated below.

ASSIGN <name> TO DATASET <dataset name>

example:

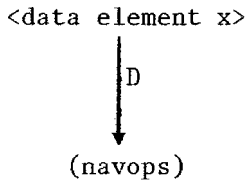
IF THERE IS an ELEMENT [x] WHOSE type IS "navy"
THEN ASSIGN [x] TO DATASET navops

ASSIGN <name> TO RULESET <ruleset name>

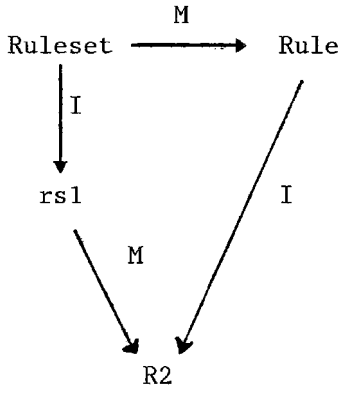
example:

```
IF   THERE IS a RULE [x] WHOSE certainty IS > .5
THEN ASSIGN [x] TO RULESET goodrules
```

Dataset definitions add names to the value of the DATASET attribute (D) associated with each data element, e.g., "ASSIGN x TO DATASET navops" sets up the link:



where D is a system attribute. However, in the case of ruleset definitions the user will perceive a hierarchy of I and M links as illustrated below.



Thus the user will be able to write rules that make use of inheritance and membership properties with regard to rule characteristics, e.g.,

```
IF   THERE IS a RULE [x] THAT IS NOT a MEMBER OF a RULESET
THEN ASSIGN [x] TO RULESET rs1
```

V. USER SUPPORT ENVIRONMENT

The user support facilities of ROSIE are intended to help the user cope with the special problems of large, rich models. Models of real-world interest may be expected to involve large numbers of rules and data elements--far too many for the model builder or user to comprehend in detail. While user support issues are important in the design of any computing system, they are especially critical in large, complex systems intended for use by non-programmers. Hence, considerable effort has gone into planning effective user support facilities for ROSIE--to provide a friendly and helpful environment for the model builder.

In this section, we outline three classes of key user support facilities in functional terms: top-level interface, editing, and model analysis.

THE USER'S TOP-LEVEL VIEW OF THE SYSTEM

Before the user can approach the substantive tasks of heuristic modeling--building data and rule elements--he must be able to invoke ROSIE from the host operating system and correctly interact with ROSIE. To aid the new user learning about the varied ROSIE features the system will incorporate a tutorial mechanism capable of describing the features and demonstrating how to use them. Thus a new user will be able to interact effectively with ROSIE even if he has a minimal understanding of the basic concepts underlying the ROSIE design.

The user must exercise control over system options and features. Thus, the system must have a set of commands that permit the user to control modes, options, file loading, running, interruption of running, resumption of running, trace setting, debugging, and the verbosity with which the system describes its own behavior. The model for these features is the latest implementation of RITA; these RITA functions will be included in the initial design for ROSIE.

There is one important top-level command, not present in RITA, that gives the user the ability to activate a particular set of rules and/or data elements from the command level; e.g.,

```
USE edit-rules;
```

This gives the user an easy way to isolate for execution a set of editing rules, or correctness-checking rules (see below), that are embedded within a user model.

The commands associated with these facilities can be typed directly at the user's terminal, or they can be embedded in loadable ROSIE files to simplify subsequent system initialization.

Also provided is the RITA concept of immediate rules--rules that are entered from the user's terminal and are executed at once. These rules differ from system commands in that they dynamically interact with the elements of the user's current model; e.g.,

```
IF THERE IS A carrier [c] WHOSE readiness IS low
THEN FOR ALL CASES DEACTIVATE [c];
```

Immediate rules differ from ordinary rules in that they do not become a part of the universe of rules within a user model; they are executed at once and then discarded.

Special editing capabilities are described below which further enhance the user's top-level control of the system.

EDITING FUNCTIONS

The principal tool that the model-builder will use to create and modify elements of his model (rule and data elements) is a text editor. In this role, the editor will carry much of the burden of interaction between the user and ROSIE. The functional properties of the editor must be designed to gracefully and unobtrusively assist the user in his work; those described in the following paragraphs are suggested by a broad sampling of user experiences with the RITA system.

The main function of the editing facilities is to facilitate the manual creation or change of rule and data elements. (Rules and data can also be created as a result of rule actions). To support this, it is desirable to provide a sophisticated prompting facility. Prompting should be optional, and should be driven by user-specified templates for the most common kinds of structures in the user's model. The goal of prompting is to save the user from routine typing and to reduce the likelihood of typographical errors.

Access to the editor can be invoked manually whenever the model is quiescent, so that rule and data elements can be edited. As an aid to debugging, an existing execution context can be preserved while manual editing occurs; this will allow model execution to be halted for editing and then resumed with no loss of current states, bindings, etc.

In addition to manual editing, there are two ways in which the user can build editing aids into the model itself using ROSIE language facilities. First, he can use rules to locate data in the model that needs editing, and package these materials for later manual editing; e.g.,

```
IF THERE IS A RULE [r]
WHOSE conditions CONTAIN {'carrier'}
THEN FOR ALL CASES SEND [r] TO edit-file;
```

will gather up all rules which mention carriers in their left-hand sides and send copies of them to a file. This file can later be edited manually and the modified contents returned to the model.

Extending this approach, the user can build rules that actually edit other rule or data elements in the model; e.g.,

```
CREATE CONCEPT tf-subs;

IF THERE IS A tf-escort [tfe]
        WHOSE INSTANCE IS submarine [s]
THEN FOR ALL CASES REMOVE [s] FROM [tfe]
AND INCLUDE [s] IN tf-subs;
```

These rules carry out a systematic change in the user's taxonomy

of submarines, moving those formerly categorized as task-force escort vessels into a new category called "tf-subs." If the number of affected elements is large, then this rule would save the user substantial manual editing labor while eliminating the possibility of typing errors.

To help the system protect privileged data fields from editing (e.g., the internal object ID field) and as an adjunct to a fairly rich prompting facility, we plan to include partial syntax checking on rules and data so that at least superficial syntax checks can be done before the material is released from the editor, saving time and computing resources.

As support for interactive use of the language, the most recent lines typed by the user at his terminal will be captured in a transparent manner. If one or more of these lines proves to be erroneous, resulting in rejection by the system, the user will be able to edit the offending line(s) and resubmit them instead of having to retype the entire sequence. Once again, this should save typing and soften the adverse effects of simple typographical errors. This facility will apply to all interactive inputs, e.g., commands, immediate rules, or prompted responses.

MODEL ANALYSIS

An important question for any model builder is whether the behavior of the model, in the most general sense, accords with his expectations and needs. Where this is in doubt, the user will want to iterate through cycles of analysis, testing and modification in the hope of arriving at a state of the model that satisfies his goals. This process of model analysis is ordinarily easier for small, sparse models for which the user is able to maintain a more or less complete mental image. For large, rich models, which ROSIE is intended to support, we recognize the importance of supplementing the model builder's intuition with specific tools to assist in the analysis process. In the following subsections we describe three sets of such tools, each of which addresses an important class of analysis problems: consistency in the model, inference with uncertainty, and explanation of results.

Consistency Among Rules and Data Elements:

An obvious source of anomalous behavior in a model is the presence of collections of rule elements and/or data elements whose members are inconsistent with one another. Here are some simple examples of blatant inconsistencies:

```
(A)     OBJECT carrier,
        NAME     Enterprise,
        LOCATION Pacific,
        ...      ;
```

```
OBJECT carrier,
NAME    Enterprise,
LOCATION  Atlantic,
...     ;
```

(B)
```
IF THERE IS A red-sub
        WHOSE range IS  LESS THAN 3
THEN USE RULESET antiplane ;
```

```
IF THERE IS A red-sub
        WHOSE range IS LESS THAN 3
THEN USE RULESET antisub ;
```

These cases could have reasonably arisen as a result of clerical error or carelessness in entering new material into the model, either directly from the users terminal or from loadable ROSIE files. Or they might arise from rule-based

What can be done about this problem of consistency? The formal issues in evaluating consistency among the elements of complex models can be very deep; there is, in fact, little hope of providing a comprehensive automated solution to the detection or correction of consistency faults. Instead, the system will include approaches to helping the user identify collections of data elements and rules that may embody consistency defects, as well as other sources of faulty behavior in the model. The primary burden of recognizing the defects themselves, and of repairing them, rests with the model builder or user. He has superior human pattern-recognition talents, and may be presumed to possess unique competence to make such judgments of his own

model. The goal of system design in this area is to provide good tools for the user to help him in this task.

Two kinds of tools will be made available, each focusing on the identification of "families" of rules and/or data elements whose properties may involve consistency or correctness issues. Both rest on the hypothesis that consistency defects of the more tractable kinds are likely to involve collections of rules and data elements with high internal similarity. The members of a family of rules would share key LHS and/or RHS elements; the members of a family of data elements would share attributes and/or values.

Simple similarity metrics can be used to construct procedures that recognize similarities among sets of rules. At the "fulcrum" of family discovery, the user will either directly supply examples of key shared materials, or will point to existing rules and/or data-elements that embody them; the system procedures will then collect the members of the implicitly defined family from among the elements of the model, and organize them to simplify the user's review. While the design of these procedures will lead them to err on the side of overinclusiveness, the user will be given control of the similarity threshold employed so he can limit the size of generated families.

In addition to the built-in system procedures just described, the user may often be able to create his own procedures for reviewing portions of the model. As in the

editing situation, he may create and invoke rulesets whose function is to identify and collect families of related data elements and rules for review.

Reasoning in the Presence of Uncertainty:

Unlike a system of classical mathematical or logical inference in which all premises and inference rules are assumed perfectly correct and reliable, the elements of a heuristic inference model may differ substantially from one another in reliability or certainty. Some rules and facts will enjoy the user's full confidence; others will have a more questionable status. Also, the user's estimate of particular facts and rules will change with growing knowledge and experience.

The main problem this situation poses for the user of a heuristic model is how to estimate the reliability of inferences and predictions which are based upon uncertain information and transformations. The developers of RITA chose to leave this issue entirely to the user as a way of avoiding difficult problems of implementation in a minicomputer environment. The most common approach, among systems which attempt to solve this problem (Shortliffe, 1976) is to:

- let the user express his estimate of the reliability of facts and rules on one or more numerical scales, and

- provide built-in functions which compute similar estimates for new inferences on the basis of the estimates of the facts and rules used to reach them.

But this approach has itself given rise to two new problems. The first concerns the form of the certainty-combining functions to be embedded in the system; it is still a matter for dispute which (if any) of several candidate functions is the theoretically 'correct' or 'best' one. The second problem is simply that none of the candidate functions has met with uniform user satisfaction; users express doubts about the confidence estimates which the system assigns to new inferences--they are irregularly higher or lower than the user himself would like to assign to the same conclusions, sometimes dramatically so.

The approach adopted for ROSIE is based on two hypotheses:

- the heuristic model builder and users of such models need help in assessing the strength of the model's inferences and predictions, but
- present understanding of the logical and psychological bases of heuristic inference is too weak to yield a comprehensive, fully automated solution which users should accept.

From these we conclude that the most useful strategy is to provide system support for the kind of inference validation that people routinely employ in coping with the uncertainties of heuristic reasoning in everyday life: a careful review of the evidence. Hence, while ROSIE invites the user to assign

"certainty factors" to his rules and facts, the system will not routinely apply special functions for combining these in evaluating new inferences; instead, facilities are provided for locating and reviewing the facts and rules used in reaching them. The final assignment of new certainty factors is then left to the user. However, there will be a few certainty combining packages built into ROSIE for use by sophisticated users who understand their effects and implications. For example, we will supply one very simple yet useful certainty combining function that works as follows. All new data produced will have a certainty equal to the minimum certainty factor (over both rules and data) used to produce it.

Certainty factors (CFs) are expressed as numbers, and the user can employ whatever kind of numerical scale the system can support for this purpose. It may be desirable to provide mapping from words representing different degrees of certainty (e.g., HIGH, MEDIUM, LOW) into corresponding numerical values--either point values or interval values. The user will be able to assign a CF to each rule and to the value of every attribute of data elements. For the time being it is assumed that, where the user does not supply a CF, the default CF will be presumed to be unity, or the highest value representing complete certainty. A single user-defined CF scale is used for both rules and data elements. In addition to their primary role in inference evaluation, it may be that CFs, like other components of rules and data elements, can play a role in conflict resolution; it is

too early to make a judgement on this issue.

The approach to validating conclusions is to assist the user in reviewing the chain of reasoning involved in reaching the conclusions, with particular attention to the weaker premises and rules. A running history of the system's actions will be maintained in a history file; the contents of this file will be used to support post-mortem traces and other debugging functions as well as reviews of inference. ROSIE will contain tools capable of searching this history file, collecting the facts and rules underlying particular inferences, and organizing these materials for the model builder's use.

When the model is run, the user will have the option of providing a cutoff point or threshold CF value. The threshold will limit the scope of the data or rules to be considered in the calculation, as only items with CF values equal to or greater than the threshold will be used in the calculation. Thus a threshold of .8 would refer to "all data elements and rules with a CF of .8 or above." If the user prefers to think in terms of more abstract CFs the system will give him a CF test to calibrate his certainty and will map it into a set of linguistic terms, e.g., CERTAIN, HIGH, MEDIUM, LOW. He will then use these values in conversing with the system, although ROSIE will internally use standard numerical values.

An inference will be made using all rules and data above the threshold value (which has a default value of 0). If no certainty combining package is specified, all new data produced

will have a default certainty of 1, as illustrated in the example below.

DATA

d1: .8 d2: .9 d3: .6 d4: 1.0

RULES

```
                              .9
r1:   c1 & c2 & c3  --->  a1

                              .8
r2:   c3 & c4 & c5  --->  a1

                              .6
r3:       c4 & c6   --->  a2
```

If the threshold value is .8 then only d1, d2, d4, r1 and r2 would be used in the calculation. If r1 was properly instantiated by d1, d2, and d4 then a1 would be added to the data with a default CF of 1, and the process would continue.

The user will be able to ascertain the true validity of an inference by querying the system about the chain of reasoning used to reach the decision. For example, he might choose to examine:

- data and rules by CF (a display of the rules and data involved, ordered by ascending/descending CFs)
- weak links (rules/facts with CFs below some user-specified threshold)
- initial data (initial rules/facts used in the chain)
- intermediate facts (new attribute values created in the course of inference)

If he lacks confidence in the decision reached by the system he can change the CFs on the rule or data elements, or change the CF threshold value, and run the model again, repeating this until he obtains a decision he trusts.

Explanation of Model Behavior:

The work of gaining insight into the behavior of a complex model has static and dynamic aspects. In the preceding sections on editing facilities and consistency checking tools we have outlined the ways in which the ROSIE system can help the user review and modify the component parts of a static model--the rules and data elements that make it up. As a model runs, its rules and data elements interact dynamically with one another and with the system's monitor. The user's concern with heuristic reasoning in the presence of uncertainty deals with one facet of the dynamic interaction among the rules and data elements. In this section, we focus on more general interactions among the rules, the data elements, and the ROSIE monitor.

Much of the information useful in explaining the behavior of the system to the user exists in the history file which the system maintains. One key approach to assisting the user to understand the system's behavior, therefore, is based on providing various specialized filters to extract from the history file information which would help the user to understand the system's actions.

One type of filter that may be defined on the history file produces an explanation of a chain of reasoning. This mechanism, because it is based upon the general history file, can be used equally well to help explain forward or backward chains of inference. At various user-controlled levels of detail, the system can exhibit chains of inference by rules, by rules and data, by rules and data with bindings, etc. In addition, the explanation facility can make use of annotations on rule and data elements, supplied as attributes by the model builder, to help the user understand sequences of system actions. If the history file internally takes the form of ROSIE data elements, then the full generality of the modeling capability can be applied to it for scanning and other similar activities.

Often, the user will require information about the behavior of the system which goes beyond inference schemas. He may want to trace all rules which actually fire, those rules whose left-hand sides were true, those rules whose left-hand sides were merely tested, those which were retrieved for a test. The user may be interested in the reason why the left-hand side was retrieved but failed to be tested, or he may want to know the criterion that was applied to exclude this rule during conflict resolution. Similar concerns may apply for the data elements: which elements have their values tested, which are members of an instance set, or which have their values set. For these purposes, ROSIE will include mechanisms for tracing the system's actions at various levels of detail.

The hierarchy among the levels of tracings can be tentatively defined as follows. At the lowest level the user will be told when a designated rule fires, at a higher level, when its left-hand side evaluates to true, and when it fires; at a still higher level, when its left-hand side is being tested, or when it evaluates true, or when it fires; at a higher level still, when the rule has been gathered in the search for eligible left-hand sides, or when it is tested, or is true, or fires. And at the highest level of all, the level of greatest detail, the system will be asked to tell 'everything' about system actions affecting the rule (or data element).

The ROSIE system will construct concise English-like descriptions of the system's behavior with respect to the designated elements, so as to avoid imposing on the user the burden of remembering in detail the functions of the monitor: conflict resolution, search strategy, and testing strategies.

The material emitted by the system in response to trace or other explanatory commands can be directed by the user either to the user's terminal (for immediate viewing) or to a file (for later use) or both.

REFERENCES

Anderson, R. H., and J. J. Gillogly, Rand Intelligent Terminal Agent (RITA): Design Philosophy, R-1809-ARPA, The Rand Corporation, Santa Monica, California, 1976.

Anderson, R. H., M. Gallegos, J. J. Gillogly, R. B. Greenberg, and R. Villanueva, RITA Reference Manual, R-1808-ARPA, The Rand Corporation, Santa Monica, California, 1977.

Bobrow, D. G., and T. Winograd, "An Overview of KRL, A Knowledge Representation Language," Journal of Cognitive Science, 1977, 1, pp. 3-46.

Charniak, E., "Organization and Inference in a Frame-like System of Common Knowledge," Proc. TINLAP, Cambridge, Massachusetts, June 1975, pp. 46-55.

Havens, W. S., A Computational Model for Frame Systems, Ph.D. Thesis, Department of Computer Science, UBC, Vancouver, B.C., 1978.

Hayes-Roth, F., and D. J. Mostow, "An Automatically Compilable Recognition Network for Structured Patterns," Proceedings of the Fourth International Joint Conference on Artificial Intelligence, Tbilisi, USSR, 1975, pp. 246-251.

Hayes-Roth, F., D. A. Waterman, and D. Lenat, "Principles of Pattern-directed Inference Systems," Pattern-directed Inference Systems, Waterman, D. A., and F. Hayes-Roth (eds.), Academic Press, New York, 1978.

Hewitt, C., "Procedural Embedding of Knowledge in PLANNER," Proceedings of the Second International Joint Conference on Artificial Intelligence, Brit. Comput. Soc., London, 1971, pp. 167-184.

Hewitt, C., Description and Theoretical Analysis (Using Schemata) of Planner: A Language for Proving Theorems and Manipulating Models in Robots, TR-258, MIT AI Lab., Cambridge, Massachusetts, 1972.

Lenat, D., AM: An Artificial Intelligence Approach to Discovery in Mathematics as Heuristic Search, SAIL AIM-286, Stanford Artificial Intelligence Laboratory, Stanford, California, 1976. Jointly issued as Computer Science Department Report No. STAN-CS-76-570.

Lenat, D., "Automated Theory Formation in Mathematics," Proceedings of the Fifth International Joint Conference on

Artificial Intelligence, Cambridge, Massachusetts, 1977, pp. 833-842.

Lenat, D., and G. Harris, "Designing a Rule System that Searches for Scientific Discoveries," Pattern-directed Inference Systems, Waterman, D. A., and F. Hayes-Roth (eds.), Academic Press, New York, 1978.

Martin, N., P. Friedland, J. King, and M. Stefik, "Knowledge Base Management for Experiment Planning in Molecular Genetics," Proceedings of the Fifth International Joint Conference on Artificial Intelligence, Cambridge, Massachusetts, 1977, pp. 882-887.

McCalla, G. I., An Approach to the Organization of Knowledge for the Modelling of Conversation, Technical Report 78-4, Department of Computer Science, University of British Columbia, 1978.

Minsky, M., "A Framework for Representing Knowledge," The Psychology of Computer Vision, Winston, P. H. (ed.), McGraw-Hill, New York, 1975.

Shortliffe, E. H., Computer-based Medical Consultations: MYCIN, American Elsevier, New York, 1976.

Stallman, R., and G. J. Sussman, "Forward Reasoning and Dependency-directed Backtracking in a System for Computer-aided Circuit Analysis," Memo 380, MIT AI Lab, Cambridge, Massachusetts, 1976.

Teitelman, W., Interlisp Reference Manual, Xerox Palo Alto Research Center, Palo Alto, California, 1974.

Waterman, D. A., An Introduction to Production Systems, P-5751, The Rand Corporation, Santa Monica, California, 1976.

Waterman, D. A., Rule-directed Interactive Transaction Agents: An Approach to Knowledge Acquisition, R-2171-ARPA, The Rand Corporation, Santa Monica, California, 1977.

Waterman, D. A., and B. Jenkins, Heuristic Modeling Using Rule-based Computer Systems, P-5811, The Rand Corporation, Santa Monica, California, 1977.

Waterman, D. A., and F. Hayes-Roth, "An Overview of Pattern-directed Inference Systems," Pattern-directed Inference Systems, Waterman, D. A., and F. Hayes-Roth (eds.), Academic Press, New York, 1978. (a)

Waterman, D. A., and F. Hayes-Roth, Pattern-directed Inference

Systems, Academic Press, New York, 1978. (b)

Winograd, T., "Frame Representations and the Declarative/Procedural Controversy," Representation and Understanding: Studies in Cognitive Science, Bobrow, D. G., and A. Collins (eds.), Academic Press, New York, 1975.